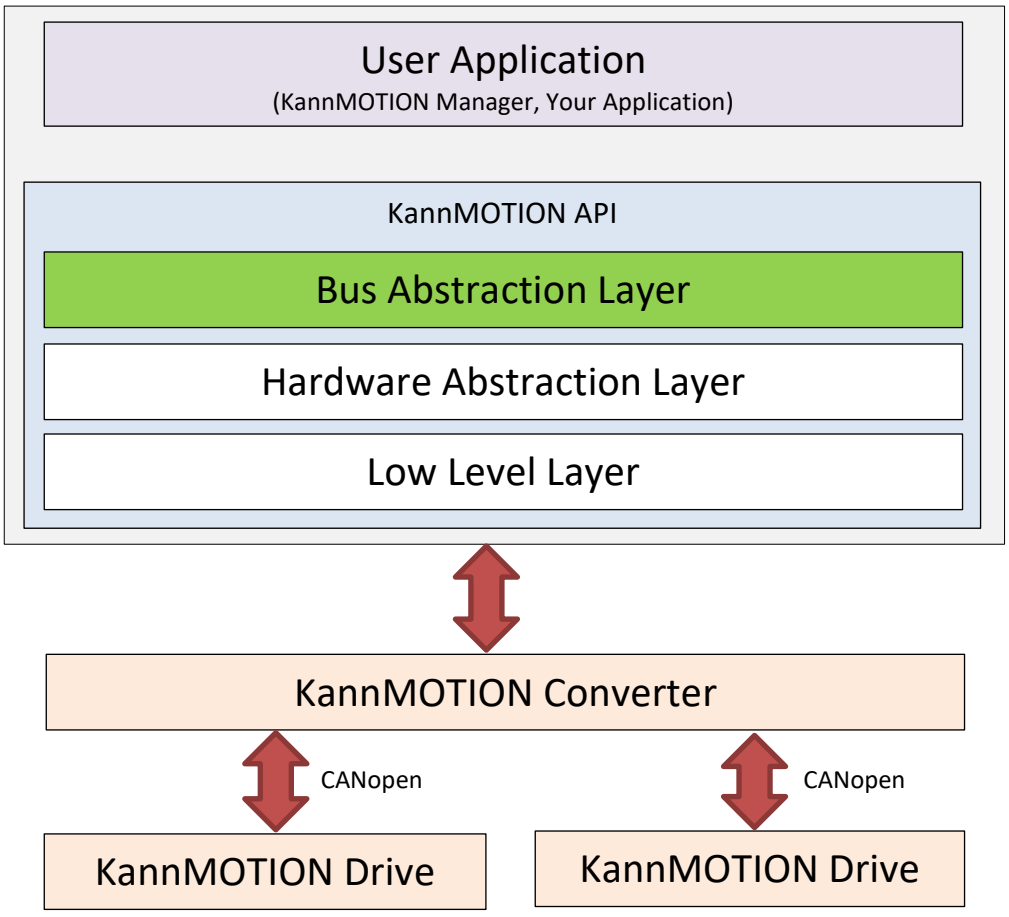
AN300047: Using Adlos Communication Bus Abstraction API [BAL]

Introduction

The KannMOTION-API serves the easy connection of KannMOTION Devices with your application.
The API is split into 3 layers, the Low Level Layer (LLL), the Hardware Abstraction Layer (HAL) and the Bus Abstraction Layer (BAL).

This particular document introduces the user to the BAL-API. The BAL library expands the HAL to search and communicate with CANopen drives via the CANopen converter.

Common Infos



⚠	The BAL is ONLY available as a .NET Core library.
⚠	This application note is for BAL with version >= 1.0.0.0

API

With this API you can search and connect devices, read information and send commands. With the **SerialDriver**-Object (same as SerialCommunicator-Object in HAL), the **CANOpenDriver**-Object or the **Converter**-Object, you can communicate with your KannMOTION device. Below a quick overview.

```
namespace Adlos.KM.Framework.Interfaces.Devices.Serial
{
    public interface IConverter : ISerialCommunicator, ICommunicator<int>, ICommunicator, IArticle,
                                  IDisposable, IComparable<ICommunicator<int>>, IEnumerable
    {
        int Count { get; }
        bool UpdateDrives();
    }
}
```

```
namespace Adlos.KM.Framework.Interfaces.Devices.Bus.CANOpen
{
    public interface ICANOpenDriver : IBusCommunicator<byte>, ICommunicator<byte>, ICommunicator,
                                      IArticle, IDisposable, IComparable<ICommunicator<byte>>
    {
        IDictionary<int, ISDOTable> SDODictionary { get; }
        IDictionary<int, IPDORXTable> PDORXDictionary { get; }
        IDictionary<int, IPDOTXTable> PDOTXDictionary { get; }
    }
}
```

Usage

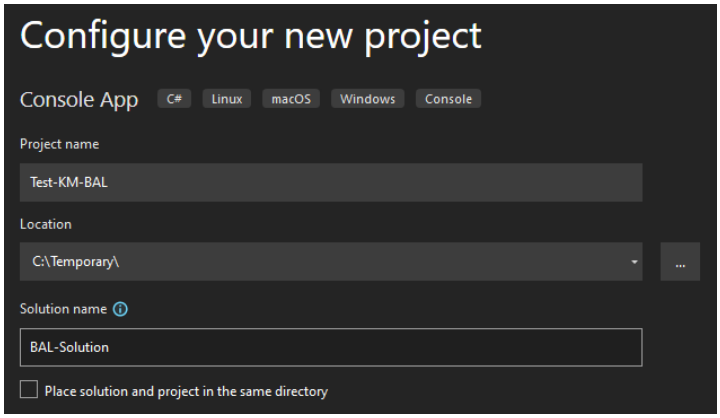
In this app note it is described, how to use the BAL DLL in a Visual Studio Project.
For this, we create a new Visual Studio Project (Console App .NET Core) to call some basic commands with API methods.
To use the complete BAL DLL, some other DLLs are needed too. Following the needed DLLs. Version number of used DLL in this examples are shown in brackets.

- Adlos.Formulator.dll (v1.0.0.1)
- Adlos.Framework.dll (v0.0.2.0)
- Adlos.KM.Framework.dll (v1.0.0.4)
- Adlos.KMAPI.BAL.dll (v1.0.4.0)
- Adlos.KMAPI.HAL.dll (v0.9.3.0)
- Adlos.KMAPI.LLL.dll (v0.9.4.15)
- Adlos.Loader.dll (v1.2.3.1)

We will create a Converter, search for CANopen devices, read some device information and call some commands (SDOs).
This example project is available as *Test-KM-BAL,* and can be opened with Visual Studio.

Additional infos for protocol, API, etc. can be found under *Documentation & Links*.

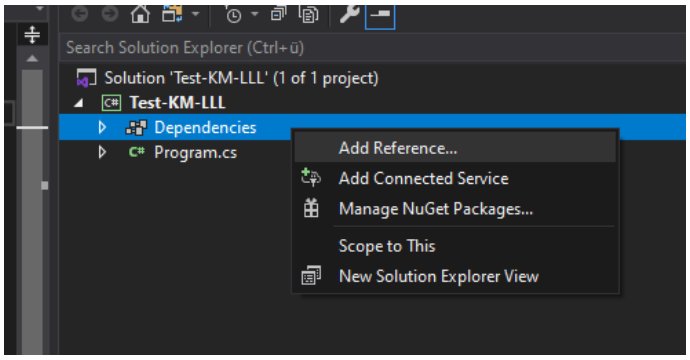
Create new Visual Studio Project



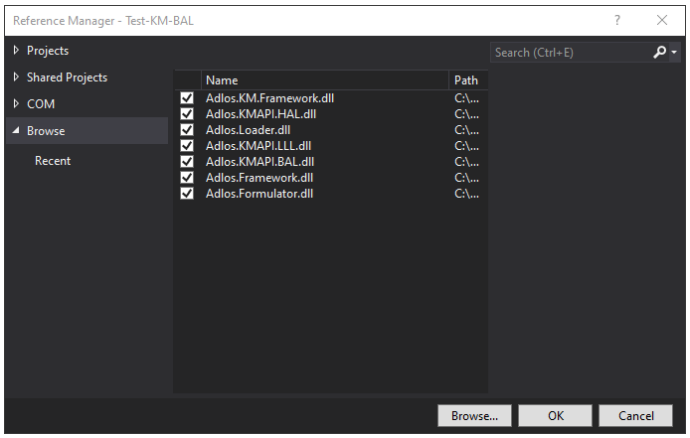
Then, click on **Next** and **Create**.

Include libraries

In your project map on the right side, right click on **Dependencies** and **Add Reference**.
In other versions of Visual Studio it can be called **Add Shared Project Reference**.



Next, go to **Browse** on the left side, click on **Browse** and select needed DLLs (as listed in Usage), then click **Add**.



Install needed NuGet

The LLL needs the *System.IO.Ports* NuGet. Again right click on **Dependencies** and click **Manage NuGet Packages...** Search for *System.IO.Ports* and install it.

Examples

Here you can find some coded examples to get to work with the BAL API library.
You can find these examples in the *Test-KM-BAL* Visual Studio Project.

Code: Using directive

Allow the use of types in the namespace so that you do not have to qualify the use of the type in that namespace. These are the ones used in the example project.

```
using Adlos.KM.Framework.Enumerations;
using Adlos.KM.Framework.Interfaces.ComObjects.CANOpen;
using Adlos.KM.Framework.Interfaces.ComObjects.Serial;
using Adlos.KM.Framework.Interfaces.Devices.Bus.CANOpen;
using Adlos.KM.Framework.Interfaces.Devices.Serial;
using Adlos.KMAPI.BAL;
using Adlos.KMAPI.BAL.CANOpen;
```

Code: Prevent downloading

```
// Flag, to either prevent or allow downloading of most recent Communication XML needed for devices
// Has to be downloaded at least once!
BALManager.PreventDownloading = false;
```

Code: Get BAL Version

```
// Get version of HAL library
Version HalVersion = typeof(BALManager).Assembly.GetName().Version;
```

Code: Connect to a KannMOTION device

Either, you look for all available converters...

```
// Search all available KannMOTION devices
BALManager.SearchDevices(out List<ISerialDriver> drivers, out List<IConverter> converters);
```

... or you search for explicit converters with COM-Ports as parameter.

```
// Search devices on given ports
BALManager.SearchDevices(out List<ISerialDriver> drivers, out List<IConverter> converters, new
string[] { "COM4", "COM12" });
```

Next, get the converter...

```
// Get first converter
Converter converter = (Converter)converters.First();
// Get converter with on com port 4
//int comPort = 4;
```

```
   //Converter converter = (Converter)converters.Find(d => d.Identifier == comPort);
   //CANOpenConverter converter = (CANOpenConverter)converters.Find(d => d.Identifier == comPort);
```

… and search for CANopen devices attached to the converter.

```
   // Search for drives on converter
   converter.UpdateDrives();
```

Then, get the first CANopen device or with given node number.

```
   // Get first drive on converter
   ICANOpenDriver canDrive = (converter as CANOpenConverter).BusDrives.First().Value;
   // Get node 3 on converter
   //ICANOpenDriver canDrive = (converter as CANOpenConverter).BusDrives.Where(x => x.Value.Identifier
== 3).Select(x => x.Value).First();
```

Code: Get SerialCommunicator informations

The SerialCommunicator-Object holds some information about the device directly available.

```
   // Get hw article number (controller)
   string hardwareArtNumber =   $"{canDrive.ArticleNumber}.{canDrive.ArticleIndex:000}";
   // Serialnumber
   int serialNumber = (int)canDrive.SerialNumber;
   // Firmware info
   string fwArtNr = $"{canDrive.Firmware.ArticleNumber}.{canDrive.Firmware.ArticleIndex:000}";
   Version fwVersion = canDrive.Firmware.Version;
```

Code: Get settings data

```
   // Get settings information
   string positioningUnit = "";
   if (canDrive.Settings.First(s => s.KMMTag == KMMTag.PositioningUnit) is ISettingPoint sp
   && sp != null)
   {
      switch (sp.DeviceValue)
      {
         case 0: positioningUnit = "um"; break; // Mikrometer
         case 1: positioningUnit = "°"; break;  // Grad
         default: positioningUnit = "Æ"; break; // Unbekannte Einheit (Æ)
      }
   }
   else
   {
      // Unbekannte Einheit (Æ)
      positioningUnit = "Æ";
   }
```

Code: Get information from SDOs

SDOs data can be found with KMMTag, updated with Update() and if available, formated value is calculated.

```
   // Get more information from SDOs
   double systemArticleNumber = 0;
   if (canDrive.SDODictionary.Values.Any(s => s is SDOTable sdo && sdo.Any(sub => sub.KMMTag ==
KMMTag.DUArticleNumber)))
   {
      ISDOTable sdoTable = canDrive.SDODictionary.Values.First(s => s is SDOTable sdo && sdo.Any(sub
=> sub.KMMTag == KMMTag.DUArticleNumber));
      var sdoPoint = ((SDOTable)sdoTable).First(p => p.KMMTag == KMMTag.DUArticleNumber);
      if (sdoPoint.Update())
      {
         systemArticleNumber = ((sdoPoint.Value & 0xFFFFFF00)/256) + ((sdoPoint.Value & 0xFF)/1000);
      }
   }
```

More can be found with KMMTag and updated with Update(). For easy use they can be gathered in a list and being updated as wanted.

Define and populate update list. ISDOTable is an SDO, the ISDOPoint is a sub index.

```
   // Get more information from SDOs
   List<ISDOPoint> updateList = new List<ISDOPoint>();
   List<string> errors = null;
   double temperature = 0;
   double actualPosition = 0;
   double totalRuntime = 0;
   List<string> dInputs = new List<string>();
   string aInput = "";

   // Populate individual update list
   foreach (ISDOTable sdoTable in canDrive.SDODictionary.Values)
   {
      foreach(ISDOPoint sdoPoint in sdoTable.Points)
      {
         if (sdoPoint.KMMTag == KMMTag.Error
            || sdoPoint.KMMTag == KMMTag.Temperature
            || sdoPoint.KMMTag == KMMTag.Position
            || sdoPoint.KMMTag == KMMTag.TotalRuntime
            || sdoPoint.KMMTag == KMMTag.DigitalInputs
            || sdoPoint.KMMTag == KMMTag.AnalogInput)
         {
            updateList.Add(sdoPoint);
         }
      }
   }
```

Next, the SDOs are updated and can be used.

```
   // Update all needed information
   foreach (ISDOPoint point in updateList)
   {
      if (point.Update())
      {
         switch (point.KMMTag)
         {
            case KMMTag.Error:
               if (errors == null)
               {
                  errors = new List<string>();
               }

               if (point is Adlos.KMAPI.HAL.Model.DataPoints.BitfieldDataPoint bf)
               {
                  for (byte n = 0; n < (bf.Size * 8); n++)
```

```
                    {
                        if (bf[n].Item2)
                        {
                            errors.Add(bf[n].Item1);
                            errors.Add("|");
                        }
                    }
                }
                break;

            case KMMTag.Temperature:
                temperature = point.Value - 50;
                break;

            case KMMTag.Position:
                actualPosition = positioningUnit == "um" ? point.Value : (point.Value / 10);
                break;

            case KMMTag.TotalRuntime:
                totalRuntime = (double)point.Value / 3600;
                break;

            case KMMTag.DigitalInputs:
                if (point.Type is DataType.BF_8)
                {
                    byte dibf = (byte)point.Value;
                    BitArray diba = new BitArray(new byte[] { dibf });

                    for (byte n = 0; n < 8; n++)
                    {
                        if (point.List[n].Contains("DI") && !point.List[n].Contains("Ain"))
                        {
                            dInputs.Add($"{point.List[n].Replace("DI_", "")}:{(diba[n] ? "ON " :
"OFF")}");
                        }
                    }
                }
                break;
            case KMMTag.AnalogInput:
                aInput = $"{(point.Value * point.Factor):0.00} {point.Unit}";
                break;
        }
    }
    else
    {
        throw new ApplicationException("Can't update selected device!");
    }
}
```

Code: Commands

Look for and use different commands.

```
   // Rotate with -100 rpm for 2s
   canDrive.SDODictionary.Values.First(s => s.Points.Any(p => p.KMMTag ==
KMMTag.RotateCommand)).Points.First(p => p.KMMTag == KMMTag.RotateCommand).Write(-100);

   // Stop
   canDrive.SDODictionary.Values.First(s => s.Points.Any(p => p.KMMTag ==
KMMTag.RotateCommand)).Points.First(p => p.KMMTag == KMMTag.RotateCommand).Write(0);
```

More commands and how to use them.

```
   // Get commands
   ISDOPoint goToPos = canDrive.SDODictionary.Values.First(s => s.Points.Any(p => p.KMMTag ==
KMMTag.GoToPositionCommand)).Points.First(p => p.KMMTag == KMMTag.GoToPositionCommand);
   ISDOPoint homingMode =    canDrive.SDODictionary.Values.First(s => s.Points.Any(p => p.KMMTag ==
KMMTag.HomingCommand)).Points.First(p => p.KMMTag == KMMTag.HomingCommand);
   ISDOPoint homingTimeout = canDrive.SDODictionary.Values.First(s => s.Points.Any(p => p.KMMTag ==
KMMTag.HomingTimeout)).Points.First(p => p.KMMTag == KMMTag.HomingTimeout);
   ISDOPoint homingSpeed =   canDrive.SDODictionary.Values.First(s => s.Points.Any(p => p.KMMTag ==
KMMTag.HomingSpeed)).Points.First(p => p.KMMTag == KMMTag.HomingSpeed);

   // Go to position -1000
   Console.WriteLine("- gotopos -1000");
   goToPos.Write(-1000);
   Task.Delay(3000).Wait();

   // Set zero position
   Console.WriteLine("- set zero pos");
   homingMode.Write(0x00);
   Task.Delay(1000).Wait();
```

Commands (GoToPos, Rotate, etc.) can only be used on drives, as long as there is no user sequence programed. Further, commands like GoToPos, Rotate etc. are only possible with firmware versions lower than 3.2.0, because CiA 402 was introduced after, and commands can not just be run like that.

Code: Close connection

Clode the connection.

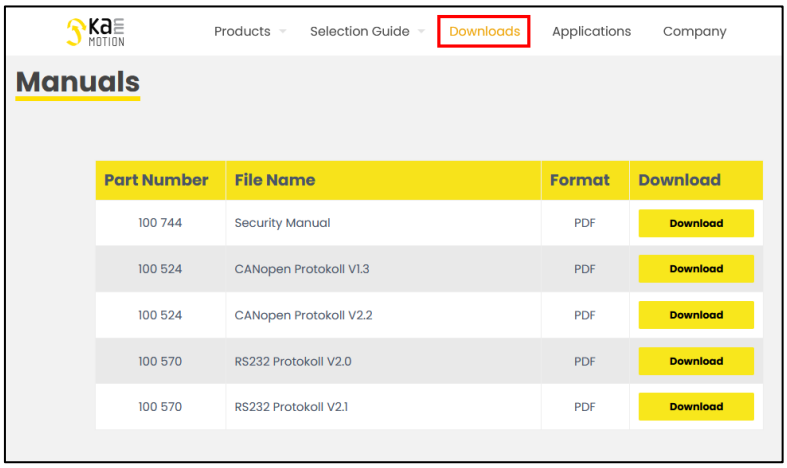
```
    // Close connection
    converter.CloseConnection();
```

Or close the connection and free all used system resources.

```
    // ...or close connection and free all used system resources
    converter.Dispose();
```

Documentation & Links

API, Protocols and additional information can be found on kannmotion.com under downloads tab.



Art. Nr.	*Description*
100570	RS232 Protocol
100524	CANopen Protocol
300054	Protocol Simplification
190073	LLL API (Low Level Layer)
190074	HAL API (Hardware Abstraction Layer)
190080	BAL API (Bus Abstraction Layer)
300045	LLL Application Note
300046	HAL Application Note
300047	BAL Application Note

Contact information

Adlos AG
Föhrenweg 14
FL-9496 Balzers

Thomas Vogt
Thomas.Vogt@adlos.com
Tel: +423 263 63 63

Countries: CH, A, LI, SK, IT
www.adlos.com

KOCO MOTION GmbH
Niedereschacher Straße 54
D-78083 Dauchingen

Olaf Kämmerling
O.Kaemmerling@kocomotion.de
Tel: +49 7720/995858-0

Countries: DE, BE, NL, LU
www.kocomotion.de